AD-A035 161 DEFENSE SYSTEMS MANAGEMENT SCHOOL FORT BELVOIR VA F/G 9/2
SOFTWARE MANAGEMENT FOR SHIPBOARD COMPUTER SYSTEMS.(U)
NOV 76 J K WILLIAMS

UNCLASSIFIED NL

1 OF 1
ADA035161

END
DATE FILMED
3 - 77

DEFENSE SYSTEMS MANAGEMENT COLLEGE

PROGRAM MANAGEMENT COURSE
INDIVIDUAL STUDY PROGRAM

SOFTWARE MANAGEMENT FOR
SHIPBOARD COMPUTER SYSTEMS

STUDY PROJECT REPORT
PMC 76-2

James Kendree Williams
Commander USN

FORT BELVOIR, VIRGINIA 22060

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) SOFTWARE MANAGEMENT FOR SHIPBOARD COMPUTER SYSTEMS,	5. TYPE OF REPORT & PERIOD COVERED (9) Study Project Report, 76-2
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Williams, James Kendree / Williams	8. CONTRACT OR GRANT NUMBER(s) (12) 94 P.
9. PERFORMING ORGANIZATION NAME AND ADDRESS DEFENSE SYSTEMS MANAGEMENT COLLEGE FT. BELVOIR, VA. 22060	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (11) Nov 76
11. CONTROLLING OFFICE NAME AND ADDRESS DEFENSE SYSTEMS MANAGEMENT COLLEGE FT. BELVOIR, VA 22060	12. REPORT DATE 76-2 13. NUMBER OF PAGES 92
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)* UNLIMITED	
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*	
18. SUPPLEMENTARY NOTES	
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)* SEE ATTACHED SHEET	
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)* SEE ATTACHED SHEET	

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

408 462 dn

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

98

DEFENSE SYSTEMS MANAGEMENT COLLEGE

STUDY TITLE: SOFTWARE MANAGEMENT FOR SHIPBOARD COMPUTER SYSTEMS

STUDY PROJECT GOALS:

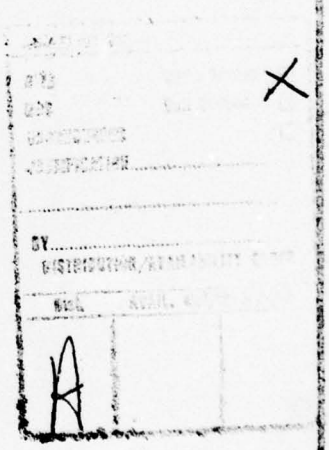
To learn present software engineering techniques as applied in Navy Ship Acquisition Project Management Offices for embedded computer systems on Naval Ships, and to consolidate and synthesize with my experience in engineering management and other computer applications.

STUDY REPORT ABSTRACT:

Current software management and software engineering techniques in all phases of embedded computer systems software development are reviewed, especially as related to Navy Ships. Impacts of DODI 5000.29 and recent developments within the Navy regarding software management are discussed, and recommendations are provided for the SHAPM and the acquisition community in general.

*Keywords

Software Management
Software Engineering
Computer Software



NAME, RANK, SERVICE	CLASS	DATE
Williams, James Kendree	PMS 76-2	November 1976

-A-

SOFTWARE MANAGEMENT

FOR

SHIPBOARD COMPUTER SYSTEMS

STUDY PROJECT REPORT

Individual Study Program

Defense Systems Management College

Program Management Course

Class 76-2

by

James Kendree Williams
CDR USN

November 1976

Study Project Advisor
Mr. Edward Speca

EXECUTIVE SUMMARY

The purpose of this paper is to examine software management for embedded computer systems in general, and then to focus on software management as applied in acquisition of Navy ships.

An integrated approach to the functions of management, the phases of the Weapons Acquisition, Ship Acquisition and software development cycles is taken, using an extensive survey of the literature regarding software management and results of interviews with software managers in several Navy Ship Acquisition Project Management offices.

Aspects of planning, cost estimating, scheduling, software systems design, programming, coding, program testing, software reliability, quality assurance, configuration management, technical performance measurement and software development organizations have been considered.

The impact of DODI 5000.29, and recent developments in the Navy regarding software engineering, such as the NAVMAT R&D Program, the tentative NAVSEA PATHWAY program, and the inclusion of software lessons learned in Ship Acquisition REEF POINTS are discussed.

A series of recommendations which merge the author's findings in embedded computer systems software problems with experience in other areas of computer application is included, with an attempt not to include recommendations already available in the literature. Briefly, recommendations were:

1. The Project Manager should not be afraid of software management, in spite of lack of expertise in the subject.

2. Recommended Management Techniques

   a. The most important effort is the planning effort.

   b. Keep software development off the critical path.

   c. Develop a Work-Breakdown-Structure-oriented return cost data base.

   d. A thumb rule for cost and time estimating.

3. Requirements and Specifications

   a. Nail down requirements early.

   b. Define and specify optimization requirements.

   c. Control attempts at solving hardware problems with software.

   d. Force the developer to keep the system user-oriented.

4. Software Awareness Training for Managers

   a. Strengthen the ECS Software Management portion of the DSMC curriculum.

   b. Use DSMC/DODCI Short Courses for PM's.

   c. Start young before they have braces on their brains.

   d. Use the DSMC "Corporate Memory".

5. Recommended Research and Development Areas

   a. Systems Specification Techniques

   b. Systems Design Languages

   c. Automation of Planning and Configuration Management paperwork.

6. Organizational

a. Stay flexible for the present.

b. Keep ECS and ADP organizations apart.

c. Further support to PATHWAY, NAVMAT R&D and NAVSEA Steering Group activities.

An interview questionnaire developed by the author from study of software management problems in DOD and industry is appended, and an extensive list of references/annotated bibliography is provided.

TABLE OF CONTENTS

SECTION I

INTRODUCTION

1. Background and Statement of the Problem

In recent years, there has been a growing concern inside and outside the Government for the rising costs of computer software. Fueled by the continued exponential decreases in the costs of raw computing power and the lagging pace of improvements in software engineering, software costs are absorbing an ever-increasing proportion of the dollars spent in acquisition of digital systems - be they commercial ADP systems or weapons systems with embedded computers. Figure 1 has been reproduced so many times, in so many places, that its origin has been completely obliterated, but no one questions its essential accuracy.

Growing costs have not been the only reason for concern about software. Poor quality and low reliability have plagued DOD components, other Government agencies and all areas of the private sector which use computers to any large extent.

Within the DOD, several studies have been undertaken [P2, P44, P45], and more are underway to determine what can be done to improve the situation. As evidence of top DOD management concern, the October 1975 issue of the Defense Management Journal adopted software management as its major topic. The article by Mr. De Roze [P15] was a precursor of a new DOD Instruction, DODI 5000.29: Management of Computer Resources in Major Defense Systems [D5]. This instruction establishes policy for the management and control of computer resources during the development,

acquisition, deployment and support of major Defense Systems. It has direct impact on nearly every Ship Acquisition Project Manager (SHAPM) of the Naval Sea Systems Command (NAVSEA), as almost all ship acquisition projects are designated major systems under the parameters of DOD Instruction 5000.1 [D1].

Some of the problems DODI 5000.29 is attacking are illustrated by the following quotations:

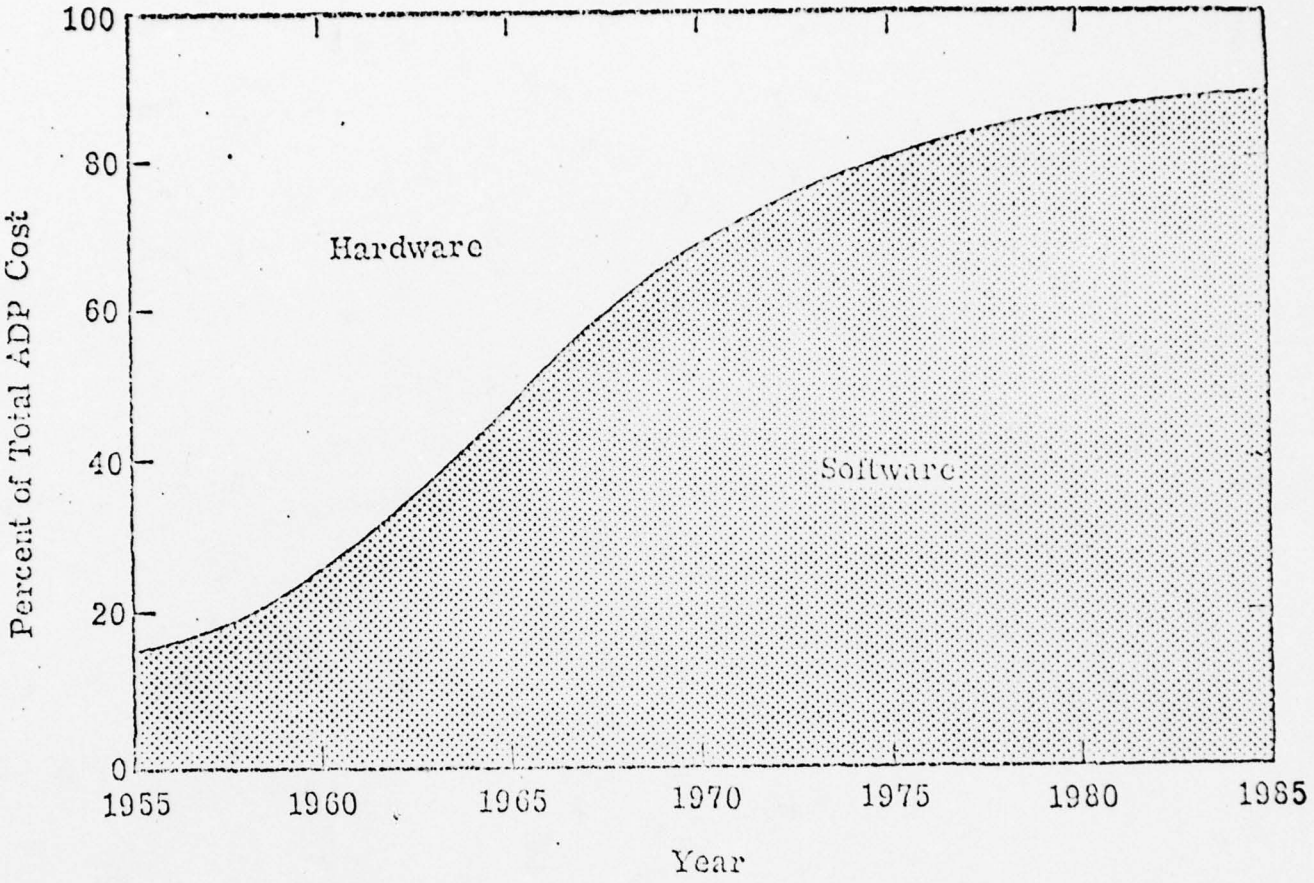
> Whereas the hardware development was for the most part scheduled, monitored, tested and regularly evaluated, the software development was not . . . .
>
> No standard procedure seems to be available within OSD for orderly testing of software items . . . .
>
> The increased percentage of development cost introduced by software makes the establishment of a suitable procedure a matter of utmost importance [P13] . . . .
>
> Documents pertaining to S/W acquisition to be used as guides by Project Managers during the development of a system are insufficient. In most cases, the available regulations, pamphlets and standards pertain to only hardware and do not address software. Therefore, the Project Manager must use his own judgement to adapt these hardware documents to software [112:20].[1] . . . .
>
> The main integrator of the ships combat system was the man interface in the Combat Information Center; with voice communication between individual systems....

[1] This notation will be used throughout the report for sources of quotations and major references. The first number is the source listed in the bibliography. The second number is the page in the reference.

> . . . reaction time from detection to release of weapon has to be drastically reduced.
>
> . . . to accomplish the reduction in reaction time, the man interface between systems has to be minimized . . . a computer-to-computer integrated combat system is required. The interface control of digital message traffic between individual systems becomes a major problem in the design and configuration management of such a combat system. The Navy has been criticized greatly for its management (or lack of) in this area. A recent example was the CGN36, USS CALIFORNIA [19:12].

References T1, T12, P12, P16, P17, P51 and J12 abound with similar horror stories from other DOD components, Government agencies and industry.

FIGURE 1



Hardware/Software Cost Trends

2. Project Goals

The goals of this project were to learn what software management or software engineering techniques exist, which ones are being used in NAVSEA SHAPM offices for shipboard embedded computer systems (ECS), whether these methods could be consolidated and synthesized with my own experience in Engineering Management, scientific computer applications, industrial computer applications and large-scale management information system development, and if such a synthesis would be beneficial.

Based on personal experience in other areas of digital computer applications, I had developed a set of beliefs about management of software development projects which I felt would probably carry over into embedded computer system (ECS) development, and I decided to use this project to compare a spectrum of software management views to see if these assumptions could be confirmed or denied and, in the process, to upgrade my knowledge regarding tactical computer systems.

Further, undertaking this project would give me an opportunity to familiarize myself with people in the Naval Sea Systems Command and elsewhere in DOD, who are concerned about software engineering and management, either through personal contact or through the literature, which would better prepare me when I report to NAVSEA for duty.

Specifically, the project would determine:

A. What problems exist in ECS software management, and are they similar to my past experiences?

B. Are there any proposed solutions that I could make the SHAPM aware of, or vice versa?

C. Could I confirm the thesis that software management is conceptually no different from any other form of developmental engineering management, and, therefore, well known systems engineering and engineering management techniques are transferable?

D. What effect does DODI 5000.29 have on the way the SHAPM must operate? What current plans and programs are in effect at the Naval Material Command (NAVMAT) and NAVSEA levels?

E. What research is in progress in the software engineering discipline, and could I recommend any further efforts?

3. Study Project Methodology and Scope

The study methods and data sources to be used were to examine the literature on software engineering/software management, including DOD, OPNAV, SECNAV, NAVMAT and NAVSEA directives [D1-D 10]; committee reports and study groups [P2, P44, P45]; procedures of commercial software houses (via industry journals) [P6, P7, P16, P17, P26-31, etc.]; and documentation from other agencies and components (i.e., NASA, GSA, Army, Air Force, etc.) [D11-D21, D23, P46, P47].

In combination, to review engineering management literature [T7, T30, P38] and some of the project management techniques espoused by the Defense Systems Management College [P1] and attempt to confirm similarities in concept or approach.

Further, to develop a structured interview questionnaire which addressed problems identified by the literature survey or suspected from experience with systems other than Shipboard ECS, and conduct interviews with software management personnel from several NAVSEA SHAPM's. In cases where the SHAPM was not the principal software management agent, I had hoped to continue the interviews at the Participating Resource Manager (PARM) level, and possibly at the Design Agent or Development Agent level; but, unfortunately, time did not permit.

In an effort to collect the maximum amount of Shipboard ECS data, especially in terms of knotty problems encountered and lessons learned, and in concurrence with the non-attribution policy of the Defense Systems Management College, neither the persons interviewed nor the SHAPM to which they are attached will be identified. Due to the wealth of

information in the published literature which identifies basically the same problems and recommends the same solutions, this omission of attribution is not considered deleterious. In fact, one conclusion has been that everybody seems to know what is wrong, and is in basic agreement as to how to correct it -- but, until recently, no one appeared to be doing much about it.

For completeness, a copy of the questionnaire is provided in Appendix A.

The final phase of the study will combine, classify and analyze the data, attempting to determine similarities in methods, successes, failures, lessons learned, and from the above, develop summaries, conclusions and recommendations.

4. Organization of the Report

The remainder of the body of the study will be presented in four sections. Section II discusses the method of organizing and arranging the data collected from the interviews and literature search by integrating the software development cycle into the various other cycles of development and acquisition, and including the functions of management. Sections III and IV contain the collected data and analyses resulting therefrom. Section V contains the author's conclusions and recommendations.

SECTION II

APPROACH AND METHOD OF ORGANIZATION

1. The Weapons Systems Acquisition Cycle

In 1971, a Department of Defense directive was issued [D1] which redefined the way in which major weapons systems would be acquired. With its amplifying and implementing directives [D2, D4], a phased acquisition cycle composed of conceptual, validation, full-scale development, production and deployment phases, with scheduled program decision points between the phases, has resulted.

Further, a Decision Coordinating Paper (DCP), and reviews by a Defense Systems Acquisition Review Council (DSARC) were established, and the components were directed to establish Program Managers with broad responsibility, authority and accountability for carrying out designated major programs.

In April 1976, the growing concern for software problems in Weapons Systems Acquisition Management (WSAM) resulted in another amplifying directive [D5] concerning management of computer resources in major defense systems.

The basic policy is stated in paragraph V.A.2.

> Computer resources in Defense systems must be managed as elements or subsystems of major importance during conceptual, validation, full-scale development, production, deployment, and support phases of the life cycle, with particular emphasis on computer software and its integration with the surrounding hardware...

As well as provisions affecting the DOD components in general, there are certain provisions which will have a direct impact on the Program Manager (PM) or, in the case of NAVSEA, the SHAPM. They are:

A. Requirements Validation and Risk Analysis

The SHAPM is now required to validate the computer resource requirements, including software, risk analyses, planning, preliminary design, security, interface control, and integration methodology during the Concept Formulation and Validation phases, prior to DSARC II. The analysis must compare the operational requirements to the planned computer resources, and include the above factors plus life cycle system planning. Software engineering elements must be major considerations in the initial design, and the risk areas, plus a plan for their resolution, must be scheduled in the DCP.

In essence, this means the SHAPM must be prepared to address ECS hardware and software aspects at all DSARC reviews.

B. Configuration Management of Computer Resources

Formal configuration management has been invoked on software as well as hardware, meaning software elements will be specified and treated as configuration items.

C. Computer Resource Life Cycle Planning

A computer resource plan is required prior to DSARC II, to identify important acquisition and life cycle planning factors.

D. Support Software Deliverables

Software support items, such as compilers, test data generators, etc. will be specified as deliverables, with DOD acquiring rights to their design and/or use.

E. Milestone Definition and Attainment Criteria

Specific management milestones for the life cycle development of the ECS will be used for program monitoring and control.

F. Software Language Standardization and Control

Invokes DOD-approved Higher Order Languages (HOL's) unless the PM can demonstrate that they are not cost effective or technically feasible in his particular area.

The instruction also establishes a Management Steering Committee for Embedded Computer Resources, which, among other things, will advise the DSARC on specific embedded computer resource issues related to major Defense systems.

Although the directive addresses major systems, it provides principles to be applied to systems that do not fall in the "major" category. Therefore, the SHAPM, whether he likes it or not, faces the intrusion of ECS matters, including software, in all his reporting and review procedures. Once the DSARC and, by implication, the NSARC, are involved, can inclusion in Selected Acquisition Reports (SAR) and Program Reviews be far behind?

2. The Ship Systems Development Cycle

Due to the extended length of the ship acquisition cycle, minor modifications to the weapons system cycle have been authorized [D8, P38]. The conceptual phase is called the Feasibility phase. Validation in ship acquisition is known as the Preliminary Design phase. The Production phase is broken into Detail Design and Lead Ship Construction followed by Major Production (also referred to as Follow Shop Production).

The NAVSEA instruction further delineates sub-cycles and phasing of activities by the SHAPM (Requiring Manager) and the functional managers (Participating Managers), including the Combat Systems Design function, which will encompass ECS software management. Combat Systems Design Requirements and Combat Systems Operational Design are further amplified elsewhere [D9].

3. The Software Development Cycle

Various authors break the software development cycle into as few as four to as many as nine phases, depending on the level of detail.

A. La Bolle [P21] uses the four-phased model:

(1) Requirements Analysis and Design

(2) Implementation and Component Test

(3) System Integration and Test

(4) Operations and Feedback (or Installation and Maintenance)

B. Metzger [T9] defines six phases:

(1) Definition

(2) Design

(3) Programming (including debugging)

(4) System Test

(5) Acceptance Test

(6) Installation and Turnover

C. In a European commercial software enterprise, Hice, Turner and Cashwell [T6] use seven phases:

(1) Definition Study

(2) Preliminary Design

(3) Detail Design

(4) Program and Human Job Development

(5) Testing

(6) Data Conversion and System Implementation

(7) Operations and Maintenance

This model is distinctive in that it considers as functioning elements of the system the portions of the overall system design that must be executed by humans. Step six is most prevalent in instances of hardware change or major software system conversion, and may be quite relevant in the Ship ECS acquisition process.

D. In an attempt to develop measurable milestones related to existing specifications and standards, Mathis and Willworth [P26] defined a nine-phase model:

(1) System Initiation and Definition

(2) Proposed Preparation and Negotiation

(3) Preliminary Design

(4) Preliminary Design Review

(5) Detailed (Computer Program) Design

(6) Detailed Design Review

(7) Implementation of Designs

(8) Integration and Testing

(9) Operations and Maintenance

Obviously, the major intent here was to emphasize the Design Reviews as important steps in themselves.

Any of these models are satisfactory, depending on the desires of the software manager. For the purposes of this study, the six phase model of Metgzer will be used.

4. The Functions of Management

The traditional functions of management are recognized as planning, organizing, staffing and controlling [T12:170], which are steps in the process of achieving organizational objectives through the efforts of other people. Ascani and Low [P1] have proposed modifications for Program Managers, and the Program Management functions which will be used in this study are planning; implementation; evaluation; coordination and control; and organization and staffing. The evaluation function could be either the function central to the maintenance of program balance between cost, schedule and performance, or it could be considered equivalent to various forms of testing. Here, it will be used for the latter.

5. An Integrated Approach

From the above, it is possible to develop an approach which integrates the software development cycle with the systems acquisition cycles and the functions of management. Table 1 displays the integrated approach which will be the basis for organizing the findings and analyses in the further sections of the study.

TABLE 1

MANAGEMENT FUNCTION	DEVELOPMENT PHASE		
	Software Development Cycle	Ship Acquisition Cycle	Weapons Systems Acquisition Cycle
Planning Organization and Staffing	Definition	Feasibility	Conceptual
Planning Coordination and Control	Design	Preliminary Design	Validation
Implementation Evaluation Coordination and Control	Programming System Test	Contract Design	Full Scale Development
Evaluation Coordination and Control Organization and Staffing	Acceptance Test Installation & Turnover	Detail Design/ Lead Ship Production	Production

SECTION III

PERSPECTIVES ACHIEVED

1. Planning

As in any other system development project, the Definition phase of a software project encompasses problem analysis and project planning. The problem analysis should define the problem and establish system objectives and performance criteria [T6, T9] in a requirements specification (e.g., Computer Program Development Specification [D15:60]). Project planning should result in a software management plan which spells out for the Project Manager the functional pieces of the desired product system, the organization, schedule, cost, and method of proving that the system meets the requirements. Metzger [T9:7] reports that system management problems which most often boil to the surface are:

- poor planning
- ill-defined contract
- poor planning
- unstable problem definition
- poor planning
- inexperienced management
- poor planning
- political pressures
- poor planning

This is confirmed for ECS by the MITRE [P2: Sect. 2; 3.2] and APL [P44: Sect. 2; 6.4-6.18] studies. SHAPM interviews indicated the necessity for good planning, and the difficulty of achieving it with existing aids.

Project plan outlines are provided by Metzger [T9] and Hice, et al [T6]. They include an overview, a phase or milestone plan, organization plan, test plan, change control plan, documentation plan, training plan, review and reporting plan, installation and operation plan, a resources and deliverables plan and, possibly most important, an index.

The requirements documents and software management plan constitute the equivalent of a Conceptual Baseline, and the wise manager will invoke formal configuration management at this stage to document effects of requirements or plan changes [D11-D14].

The Design phase involves designing the system and updating and refining the software management plan. It ends with a Critical Design Review (CDR), and the accepted software system design is the equivalent of an Allocated Baseline. Metzger and Hice, et al provide excellent coverage of this area. The APL and MITRE studies recommend application of the systems engineering process to systems design in this phase - in particular, structured and/or modular design [T4, P28-31].

Specific facets of the planning process pertinent to the study findings are:

A. Estimating [11, 18, P22, P43, P49]

There is such a multitude of estimating techniques for software projects that one is reminded of the reason there are so many techniques for numerical integration - if one really good one existed, everybody

would be using it instead of trying to invent another.

The choice of cost estimating methods is, basically, the same as for hardware projects -

(1) Engineering estimates - "bottom up" or "grass roots" estimates based on the sum of costs to do each elemental action in the Work Breakdown Structure (WBS).

(2) Parametric estimates - cost estimating accomplished by correlating parameters to historical costs through regression analysis to develop cost estimating relationships.

(3) Analogous estimates - analogies are drawn to known costs of similar items in prior systems, with adjustments to account for differences between the old system and the one being estimated.

(4) Cost to Cost Factoring - estimates based on correlating proportions of historical costs to each other and estimating on the basis of similarity to the item being estimated.

SHAPM interviews indicated cost estimating to be a major soft spot. The recent American Institute of Industrial Engineers Software Conference devoted an entire session to the state-of-art of software estimating [p49)] The APL [P44:6.10] and MITRE [P2:XV1] studies address lack of cost estimating capability. Findley [18]. Andres [11] and the interviewed SHAPM's recommended breakout of software in a standard WBS and collection of a data base for "corporate memory" of software return costs versus estimates. At this stage, it is hardly possible to say whether software costs consistently overrun, or whether they are consistently underestiamted.

B. SCHEDULING [T2, T13, P26, P28, D22]

Schedule quality and schedule adherence are largely dependent on the quality of the cost estimates and the associated assigned manning. Of the three standard scheduling techniques, line of balance, Gantt Charts and Networks, all the interviewed projects used some sort of networking, especially for interfacing and integration of software development key events with hardware delivery, installation and checkout.

The APL and MITRE studies are relatively silent on scheduling problems. Szweda [T13:Ch. 6] has the most complete coverage. Mathis and Willmorth [P26] and Morgan and Lightman [P28] develop schedules around milestones or key events with measurable, definite objectives such as deliveries, review completions, etc. for progress monitoring purposes.

The key element seems to be to recognize that software schedules will be quite likely to slip, and, thus, the original plan must keep software development milestones off the critical path. Similar to design engineering management, software development is not amenable to flooding with manpower to make up slippages and missed schedules [P5]. In fact, adding additional people can often make matters worse instead of better.

C. WORK BREAKDOWN STRUCTURE [D18]

Scheduling the work into functional elements is a key management tool. Further, it meshes exceptionally well with the newer structured techniques of systems design and program implementation (see Part III.1.D below).

Standards for WBS of software were inadequate or non-existent for most SHAPM's. Those who participated actively in the software management process developed their own, and it is probable that PARM's or development agents who were tasked by non-participating SHAPM's were also forced to develop their own. The APL and MITRE studies mention WBS only peripherally, in that they encourage structured programming [T4].

Development of WBS standards and techniques for software should take a high priority, because of the interface with documentation of costs for "corporate memory", the tying of WBS elements to deliverables and schedules for monitoring purposes, and the definition of boundaries for development of interfaces.

D. STRUCTURED DESIGN

Structured design and structured programming techniques are the software fad of the '70's, largely because they do show some potential for turning a practice which many regard as a black art into an engineering discipline.

Dijkstra [T4] initiated the concept for programming, and Warnier, of Honeywell-Bull in France, has produced an important book on logical design of systems which applies structured concepts to systems design. Defense [P2, P44, P45] and commercial [P20, P29, P30, P35] studies have leapt on the bandwagon.

There are several references in the bibliography for explaining top-down, structured or hierarchial techniques, so only the briefest of summaries will be provided here.

Formerly, the most common approaches were bottom-up design and programming, or top-down design and bottom-up programming. Structured

methods start at the top and work down a hierarchy of functions, similar to the "tree" structure of data processing [T8:305] or the traditional bureaucratic model organization chart [T16:328]. Each level in the hierarchy chart redefines the level above in more detail, and each node (functional block) represents a hierarchy plus input-process-output package (HIPO) [P20]. In program implementation, lower levels can be simulated for test purposes by "stubs" which accept and check input and provide properly interfaced output. This allows the integrated test phase to be concurrent with the implementation phase. Once the hierarchy is properly interfaced, the process portion can be programmed, defining lower levels, if necessary.

If this looks strangely like what is commonly known as Systems Engineering [D29], it's because that's what it is - only computer software experts and especially business ADP specialists think they invented it themselves.

The bibliography lists several papers which give simplified descriptions of the application and potential of structured, top-down methods in design, specification development, and software management [P29, P30, P35, P39].

Two of the newer projects using Shipyard ECS interviewed have specified and are using structured techniques and find it advantageous for control, interface management, testability and system integration purposes.

A draft Navy standard [D21] proposes to invoke structured methods on all future Navy software products.

E. SPECIFICATIONS

The Computer Program Development Specification (CPDS) and Computer Program Product Specification (CPPS) [D15] are the SHAPM or PARM's method of imposing requirements on the software developer. Configuration Management [D11-D14, P24] specifications have been invoked by SHAPM's/PARM's and are now required in all instances by DODI 5000.29.

The SHAPM must consider which Higher Order Language (HOL) to specify, and whether to invoke modular design, structured programming or other techniques. Also, what deliverables and documentation will be called out on the Contract Data Requirements List? Who will have the integration responsibility?

One item occurs which was not mentioned in any interviews, and received less than a passing glance in the APL, MITRE, SRWG [P2, P44, P45] and other studies - what should be optimized?

Once in the programming stage, the programmer has many optimization options - minimum memory requirements, minimum execution time, ease of maintenance, minimum development cost, minimum programmer effort, maximum elegance, etc. If given no guidance via specifications, he will opt for one of the last two. Moore [P27] and Curry [P11] provide further thoughts for software specifications.

2. Implementation

The implementation function corresponds to the programming and debugging phase of software development. This is where the detailed design is translated into executable machine language code, and the individual modules undergo unit testing (debugging) by the programmer. It is the phase in which the heaviest management interest is often placed in terms of increasing productivity; e.g., structured techniques, modularity, programmer productivity [P4], even though various studies [18, P5, P42] show that programming consumes only fifteen to twenty percent of the time and dollar resources expended on software development.

The APL and MITRE studies recommend implementation procedures such as software development tools and facilities, disciplined programming (engineering discipline vs. art) and integration and test capabilities [P44: Chap. 2] [P2: Chap. 3].

Neither the SHAPM nor the PARM will be actively involved in this effort, since the contractor or development agent will actually be performing the work. However, certain considerations do arise which should invoke the SHAPM/PARM interest, such as:

A. Hardware-Software Trade-Offs

Often, as critical milestones approach, there is a tendency to depend more and more on software flexibility to solve problems that arise in hardware development - a "we'll take care of it in the Program" attitude. Unless properly controlled through configuration management, this is inviting disaster. It is most common when a contractor has both hardware and software responsibility.

The type of hardware and its capabilities and limitations have a definite impact on software producibility and cost. For example, programming costs increase geometrically with reductions in memory allocation for a given function. Requirements for minimized execution time have a similar effect. This is not to say that such requirements should not exist - only that there is a trade-off between paying the programmer to meet the requirement versus buying more memory or a faster processor.

B. Higher Order Language

In most cases, CMS-2 has been specified for existing Combat Systems. Apparently, several dialects exist, and there may be options or trade-offs here that should concern the SHAPM/PARM. If a development agent proposes assembly or machine language, it should be resisted to the utmost.

C. Development Agent Facilities and Philosophy

In the choice of contractors for software production, the SHAPM/PARM should be concerned with what programming aids and facilities the competitors provide. For instance, are there debugging aids such as tracing, breakpoint insertion, etc. available? What is the contractor turnaround philosophy regarding development programs - on line (instant turnaround) with interaction, remote batch, or standard batch? Will he use the same computer for program development that he does dor daily processing? What priority will he establish for your project? There are many instances where debugging (unit testing) is relegated to the third shift or weekends. Obviously, this impacts elapsed time.

The management approach he will take is also of considerable import to the SHAPM/PARM. The Chief Programmer Team [P3] concept is a way of projectizing software development which appears to be quite advantageous when used in conjunction with top-down, structured approaches. The Chief Programmer has overall responsibility for the product development, and has as team members programmers, configuration controllers, documentation specialists, and, as a key member, a program librarian for control of stubs and drivers, as opposed to completed, tested code. The question the customer must ask is whether his project will be projectized or manage d by dispersion to the functional codes; and, if projectized, will there be a full-blown Chief Programmer Team established?

D. Interface Management

The SHAPM must determine very early in the development cycle whether he or the PARM will control interface management, and to what level. Some interfaces might be relegated to the developing contractor, but it is this author's contention that interfaces should be controlled under configuration management with a formal Interface Control Board, just as in the case of engineering changes. The Computer Program Interface Document (CPID) is the specifying vehicle, and should be carefully scrutinized during the CDR.

Possibly the highest risk area for an ECS is overall system integration, and the CPID/Interface control established will have a great bearing on reduction of that risk.

E. When does the Programming Start?

Practically all interviews and references agree on this question. Programming should not be allowed to start until the CPPS and CDR are complete. There is enough potential for rework even with formal baselines and strict configuration control. Programming just doesn't consume enough of the total time or cost to warrant jumping the gun without adequate (and agreed upon) design specifications.

3. Evaluation

The evaluation function encompasses integration, systems testing, and acceptance testing. This is where more than half of the software funds for a project are spent; and, in the integrated systems test, incalculable dollars and time can be lost due to delays and standby time.

Primary considerations here are test philosophy, test methods available, software quality assurance, reliability, maintainability, and acceptance criteria.

A. Testing

In the commercial world, testing often receives short shrift; but, fortunately, the DOD and Navy philosophy place a great deal of emphasis on adequate testing. For instance, land based test sites for Combat Systems are the norm [D3].

Test philosophy can be tied to program design philosophy; i.e., bottom-up testing vs. top-down testing. One successful project [I14] espouses the idea "build a little, test a little, build a little more." Another (also successful) uses top-down testing with lower levels of the structured design stubbed off [P52].

Testing consumes around 50% of the software development resources [P5], so large amounts of concern and attention are being paid to it because of the very high payback potential [P2, P25, P44, P45, T5].

(1) Hetzel [T5] provides a set of definitions which help build a conceptual framework for discussing assertions and specifications about program behavior.

a. Proof - use of axioms and rules of inference to provde logical correctness without regard to the environment.

b. Validation - demonstration of logical correctness in a given external environment.

c. Verification - use of executions to show logical correctness in the test environment.

d. Certification - official endorsement of correctness and effectiveness based on experience.

e. Performance tests - non-logical properties such as run time or resource utilization (program efficiency).

Debugging is specifically omitted, as it is a correction method, not a test method.

> When debugging is completed the program definitely solves some problem. Testing seeks to guarantee that it is the problem that was intended.
>
> F. Gruenberger

(2) Test principles are also given by Hetzel [T5:17].

a. Segmentation - separation of a large problem into smaller sub-problems and testing individually allows added conceptual clarity, economy in test cases, and testing in parallel with program development.

b. Design for testability - developing testable problem specifications, designing in testing aids, and structuring programs so they will be testable.

c. Simulation - use of simulation as a tool for controlling test inputs, providing a controlled program environment, and avoiding delay when the actual environment is not available (cf land based test sites) [P40].

d. Sampling - statistical testing. Simple, economic method if it were dependable. Fruitful research area.

e. Logical Reduction - seeking program transformations that make testing easier or more reliable (directed graphs, flow charts, etc.). Far away from practical success- still in "laboratory" stages.

e. Standardization - use of standardized code structure to simplify program analysis and proof techniques. Important as a means for simplifying the testing problem and developing a widely used and accepted methodology - however, if standard techniques do not employ a technology as good as or better than the one the programmer devises for himself, the result can be an inferior program.

f. Automation - various types of automated test tools.

(1) Test data generators

(2) Checkers (including theorem provers)

(3) Testing estimators

(4) Transformers (see logical reduction)

(5) Monitors and simulators (see simulation)

The Hetzel book and the second volume of the Software Reliability Work Group Study (SRWG) [P45] give excellent coverage of the state-of-the art of software testing.

B. Reliability

Software reliability is also receiving a great deal of attention currently. Although it is relatively easy to detect unreliable software, it was not until the SRWG [P45] meetings in May 1975 that a definition of software reliability was propounded. The SRWG defined software reliability as "the probability that the software (contained in an ECS) will satisfy the stated operational requirements for a specified time interval or a unit application in the operational environment." [P25].

Studies of software reliability are intimately connected with testing and test methodology. Three areas exist which seem to stand on their own as reliability items independent of testing. They are:

(1) Reliability measurement techniques - Schniedwind [P37] attempts to determine the equivalent of Mean Time Between Failures (MTBF) for certain versions of operational Navy Tactical Data Systems (NTDS) Software. He defines a "Mean Time Between Troubles" (MTBT) and uses statistical methods to calculate the reliability of those systems. Lipow [P23] presents a formal mathematical approach based on Bayesian methods. This is an active research field, and it is difficult to say at this time whether the calculated reliabilities reflect the real world or not.

(2) Fault Tolerant Software - This involves increasing reliability (whether it can be calculated or not) by designing in redundancies. Randell [P34] proposes a technique of dynamic error correction by use of recovery blocks containing an acceptance test specifying the conditions under which the computation is acceptable and alternate ways to perform it if it does not pass the test. Watson [T15:246] gives concrete examples of implicit and explicit redundancy for checking in time-sharing system monitors. Hecht [P19] provides an extension of Randell's model with details of recovery blocks and nested recovery blocks for spacecraft ECS. These concepts make the practice of "graceful degradation" long used in hardware systems as components fail applicable to software as well.

(3) Ease of Correction

Because working software generally continues to work until input conditions or hardware problems bring forth a hitherto undetected bug, maintainability also means something different in software engineering. This gets back to the debugging question so adroitly dodged in the discussion on testing. Greater experts than I have wrestled with quantifying (or describing) software maintainability, so I will beg off with the following questions:

Once a bug is detected (known to exist),

How hard is it to locate?

How hard is it to correct?

Can it be corrected without generating other bugs elsewhere (and how do you know whether you did or not?)?

To quote Dijkstra:

> "Program testing can be used to show the presence of bugs, but never their absence." [T4].

C. Quality Assurance

If the reader feels that software reliability and maintainability are nebulous at this stage of developing software production from an art to an engineering discipline, he should try wrestling with quality assurance.

Ridge and Johnson [T10:141] devote an entire chapter to software quality assurance which does little beyond discuss system and acceptance testing. Clapp and La Padula [P9:12] define software quality in terms of correctness, comprehensibility and reliability. Of these, comprehensibility is the only one not already discussed. The MITRE study [P2:3.11] devotes a good deal of space to quality assurance of software, but their recommendations mainly serve to point out just how little we know about the subject and where some research should be undertaken.

Brown [P6:637] provides some elements of software quality which might be used as workable quality measures. They are understandability, completeness, conciseness, portability, consistency, maintainability, testability, usability, reliability, structuredness and efficiency. He further suggests some practical methods for achieving quality and recommends serious study with emphasis on improved definitions, cost of quality and measurability.

Brown's tentative definition has more appeal to this author because of the inclusion of efficiency. For example, consider the two program segments below which calculate the sum of an array of 1000 numbers and store the result in the 1001st position:

```
      C          VERSION ONE                C     VERSION TWO
                 SUM = 0                          A(1001) = 0
                 DO 20 I = 1, 1000                DO 20 I = 1, 1000
                 SUM = SUM + ARRAY (I)            A(1001) = A(1001) +
                                                 *A(I)
           20    CONTINUE                   20    CONTINUE
                 ARRAY (1001) = SUM         C     END OF DEMO
      C          END OF DEMO
```

Which is the higher quality program? Both pass every test of Clapp/La Padula. However, if we consider Brown's added prescription of efficiency, version two requires 3001 index calculations and version one only requires 1001. The choice now is obvious.

One quality control (as opposed to quality assurance) technique that can be adopted is the use of some sort of non-conformance reporting and follow-up system. This has several advantages in that it ensures errors are not left uncorrected, allows determination of where errors are occurring, and develops a data base for determining whether certain types of error are prevalent enough to warrant application of special management effort or development of new techniques to prevent them. Further, "bug reports" and proposed fixes can be formally inducted into the Configuration Control program to be discussed later in the study.

4. Coordination and Control

This management function is spread through every phase of the software development cycle. It involves coordination of functions and personnel to maintain balance of cost, schedule and technical performance, monitoring the three program balance elements above and taking corrective action as required, and running of the Interface Control, Change Control, Baseline Control and Documentation Control portions of the Configuration Management Plan.

A. CONFIGURATION MANAGEMENT (CM) [P24, D11-D14]

This topic has been discussed in some detail in earlier sections. The Configuration Item (CI) is the software item which satisfies end use functions and is designated by the Government for CM. CM involves item definition; change control and record keeping; and reporting via Configuration Identification, Configuration Change Control and Configuration Status Accounting. Configuration Change Control is the systematic evaluation; coordination, approval of disapproval; and implementation of approved changes in the configuration of a CI after formal establishment of its baseline.

Recommending adoption of formal CM, with Software Change Control Boards, as early as possible in the software development cycle is somewhat akin to recommending Motherhood, Flag and Country; but, all interviews, studies and surveys, DOD and elsewhere, pick this out as a prime recommendation, the disregarding of which invariably leads to problems.

Mc Carthy [P24] provides a general overview of software CM, and two earlier Defense Systems Management College individual study projects [19, 114] provide many worthwhile details. Piligian, et al [P32, P33] provide historical insight.

B. COST, SCHEDULE AND PERFORMANCE MONITORING

Golub's Twelfth Law states, "Projects progress quickly until they are 90 percent complete, and then they remain 90 percent complete forever" [P18].[2]

Monitoring the three components of program balance is vital to proper management - first plan the work, then work the plan. The question is, how?

In an excellent DSMC individual study project, Bucciarelli points the way [12].

(1) For cost and schedule, once the WBS is structured so that it matches the top-down design, as recommended earlier, any acceptable Cost/Schedule Control Systems Criteria technique can be used [D24]. Comparisons of the Budgeted Cost of Work Scheduled, Budgeted Cost of Work Performed and Actual Cost of Work Performed can be analyzed as in any hardware or engineering project.

(2) Milestones can be used for schedule monitoring, if they are properly selected in advance and tied to deliverables so that a physical check on completion is possible [P26, P28].

[2]Although developed for commercial ADP systems, this and the remainder of Golub's Laws of Computerdom are so appropriate to this study that they are appended.

(3) For those who are so inclined, a matrix method can be used to calculate an overall completion percentage. Suppose we have n modules at the lowest level in the WBS, and m operations to be carried out (design, code, debug, test, document, etc.). Let w_{ij} be the percent of the total effort required on module i to carry out operation j. Then we have an nxm matrix where $\sum_{j=1}^{m} W_{ij} = 100$

If c_{ij} is the percent complete of operation j on module i, we have

$$C_i = \% \text{ complete of module } i = \sum_{j=1}^{m} C_{ij} W_{ij}$$

and if T_i is the percent of module i in the total package

$(\sum_{i=1}^{n} T_i = 100)$, the the overall percent completion is

$$P = \sum_{i=1}^{n} T_i \cdot \sum_{j=1}^{m} C_{ij} W_{ij} = \sum_{i=1}^{n} T_i C_i$$

This sort of thing can be handily programmed, and it can rapidly get out of hand if n and m get very large because of the size of the input problem (T_i and W_{ij} need only be prepared once; but <u>all</u> C_{ij} must be prepared every reporting period). PM's planning to put something on the computer because "it's easy to fix up a program for that" would be well advised not to overlook the input problem, and consider whether the value of the information is worth the cost.

(4) Technical performance measurement must be tied back to testing, QA and CM. The Preliminary and Critical Design reviews, test reports, and summaries of progress meetings can be used.

The specifications establish a baseline; thus, one can monitor the basic constructs common to any software system, comparing them to the baseline. For example, by examining the structure, to see if the system can/will be able to carry out its functions, and by examining its organization, to see if it will be "efficient". Examples of structure and organization are:

Structure

Data structures - lists, queues, matrices, trees, rings, etc.

Control structures - logic/decision flow, program structure, HIPO, etc.

Processing functions - real time, multiprocessing, multiprogrammed, interrupt servicing

Organization

Program organization - common storage, local/global, etc.

Data organization - file organization, accessing, searching, etc.

Memory management - paging, overlays, protection, dynamic storage allocation etc. [12]

Another approach to have an independent validation and verification by separate contractor of agency, such as often the case in the SHAPM/PARM and PARM/Fleet Combat Direction Systems Support Activity (FCDSSA) and SHAPM/PARM/Naval Undersea Center (NUSC) relationship. The most common monitoring method found by interviews and in the literature was regular progress review meetings at which cost, schedule and technical performance were reviewed in terms of milestones.

To summarize, software technical performance measurement during design and programming is no better developed or easier to measure than hardware technical perf ormance during full-scale development [D17:7, 11].

5. Organization and Staffing

Aspects of organization and staffing permeate the entire software development process. In terms of SHAPM or PARM, however, there are two times in the cycle when the organizational/staffing function are more critical - the definition phase, and the installation and turnover phase. In the definition phase, the Software Management Plan must be developed, requiring careful planning by the SHAPM as to how he is to organize and staff the management side of the software development. The producer, be he contractor or in-house development agent, will have his own staffing plans and problems, but they need not concern us here.

During preparation for and execution of the installation and turnover phase, the SHAPM/PARM has a vital interest in preparing the user organization to adopt the new system and be able to use it effectively.

With the DCP/DSARC requirements of DODI 5000.29, it appears that the SHAPM will have to have some software expertise in his own organization, rather than relying on the PARM for expertise. Several SHAPM's already have this capability, some more visibly than others.

Also, the Project Manager himself will have to develop enough "software awareness" that he can discuss risks, hardware-software trade-offs, and requirements at reviews with higher levels.

The APL report [P44:6.41] recommends that major PM's be staffed with personnel experienced in software systems engineering of sufficient stature and number to carry out essential management functions that cannot be delegated. This recommendation cannot be implemented without career incentives in software engineering and training in ECS software (not

normally available in Computer Science programs). The report further recommends that the Program Manager engage a Systems Engineering agent, particularly for assistance in RFP preparation, specification development (CPDS, CPPS, CPD, Computer Systems Resource Development Plan, etc.), and technical guidance during PDR and CDR. Also, in preparation for the Installation and Turnover phase, that a Software Operational Support (maintenance) agent be identified and consulted relative to the computer system life cycle, especially concerning maintenance support requirements, system integration, testing, and transfer from development to operational status.

As far as I could determine, these two recommendations are part of current NAVSEA practice, using NAVSEC, NUSC, FCDSSA and/or contractual assist.

The MITRE report [P2:3.41] limits its recommendations to establishing methods for getting and keeping weapons systems software engineers through training, career incentives and assignment stability.

Interviews concurred with the above recommendations.

In terms of installation and turnover, because lead ship construction and integrated Combat Systems tests are going on at the same time, Land Based Test Site (LBTS) are an accepted necessity. They offer a method of keeping software development off the critical path, and for crew training while the ship is under construction or has none of the Combat Systems support capabilities such as cooling water, electrical power and dry air available.

Once a commitment to a LBTS is made, the remaining question is, besides formal planned testing, how much benefit can be extracted? Examples of additional uses of test sites are simulation of problems detected by operational ships at sea and using experts on site to debug and prepare patches for radio transmission to the ship, development of access routes and interfaces required for Fleet Modernization Program alterations, crew training sites for follow ships, and test of new equipment without tying up valuable ship time.

SECTION IV

RECENT DEVELOPMENTS

1. General

Software management and software engineering problems were receiving high level Navy attention well before the issuance of DODI 5000.29. In fact, a concentrated effort became visible shortly after publication of the APL and MITRE reports.

By Fall of 1975, the Chief of Naval Operations invoked formal Configuration Management on Embedded Computer Resources software for surface ships. A Software Configuration Control Board was mandated for all projects, with life cycle support considerations to be covered by including the software maintenance agent as a participating member, and as Chairman in the case of Command and Control systems. The Ship Logistics Manager (SLM) was directed to ensure continuance of the SCCB after transition from production to operation.

An Interface Design Specification (precursor of the CPID mentioned in III.2.D and III.4.a above) was also mandated, requiring agreement between interfacing PARMS [D6]. NAVMAT implementation directives followed shortly thereafter.

Late in 1975, COMNAVSEA initiated a study of the return costs associated with software development. This study showed that costs increase exponentially as program size (number of machine lanaguage words) increased, and that they appear to increase exponentially with increased complexity (using number of targets tracked simultaneously as

a complexity measure), but with less confidence due to paucity of data. A thumb rule upper limit of forty dollars per machine language word seemed justifiable, but there was no accurate way found or proposed to estimate the number of words.

The study recommended (again) that software development cost data be broken out to the same level of detail as hardware cost data; i.e., programs, design effort, development effort, integration, and documentation, with contractual requirements for milestones with budgeted cost at each milestone, all reported under a WBS.

Further, it recommended that a "Tactical Computer Office" be formed to assist NAVSEA, SHAPM's and PARM's in cost estimating, computer resource development planning and monitoring, and to function as a continuous repository of cost and other software management data for maintenance of a corporate memory [P41]. At the time of this paper neither recommendation had been adopted officially.

On another tack, commencing in 1974, the PATHWAY [48] program was initiated as a major coordinated R&D effort within NAVSEA to provide a technology base from which confidence could be gained that shipboard installed computer programs are acquired, installed and tested, properly interlocked and synchronized with ship design, construction and post-delivery operations. Details of the program will be provided below.

Shortly after distribution of the draft R&D plan, a NAVMAT Computer Sciences R&D Council (CSRDC) [D10] was formed. All systems commands are represented on the CSRDC. The functions of the new Council are to coordinate among the systems command and validate/revise the draft R&D plan to forward a comprehensive NAVMAT position to CNO. Further, to

establish a vehicle to measure progress against the formulated plan, and (see Section 11.1) review Advance Systems Concepts, Operational Requirements, Development Proposals and Navy Development Concept Papers to assure technical balance, inter-project coordination and avoid redundant effort. The NAVSEA Steering Group provides members to the CSRDC.

The CSRDC kick-off meeting was held in September 1976, and the proposed plan is to complete validation, including SYSCOM inputs, in two-to-three months. Details on the NAVSEA input to the R&D plan are below.

As the culmination of a several month effort, a new revision to the NAVSEA Guide to Ship Acquisition, Reef Points was issued in September of 1976 [P51]. Part 11 of the new edition is entitled, "Lessons Learned in Management of Ship Acquisitions", and includes a major section on software acquisition lessons learned. Details are provided below.

Although it was a DOD-wide effort, rather than purely Navy, the workshop on Management of Software Acquisition for Embedded Computer Systems held at the Defense Systems Management College in June, 1976 should not be overlooked, and a sub-section is included below [P47].

Another Navy effort in progress is preparation and issue of a new Military Standard [D21] for Tactical Software Development. A draft was circulated on 28 June 1976 for review. It currently has no official status, but some of the proposed contents are illustrative of the thinking which has been sparked in the last two years.

If approved without major changes, it would impose top-down structured programming, limit the use of re-entrant programs, prohibit recursive programs, and limit modules to no more than fifty CMS-2 statements (CMS-2 being the invoked HOL in the standard). It would impose top-down design and encourages the developer to use some equivalent or variant of the Chief Programmer Team concept to facilitate top-down design, coding and test.

Due to the tentative nature of the specification, further analysis and comment are not warranted in this study.

2. REEF POINTS [P51]

The overall purpose of REEF POINTS is to provide, in simple, clear language, a description of the Ship Acquisition process as defined in DODD 5000.1. It is divided into three parts - Description of the Acquisition Process, Lessons Learned in Application of the Process, and a Listing of Directives being on the Process, with a synopsis of each directive. It includes Reader Information Service on selected topics, and a feedback system for incorporating new lessons learned and comments. Inputs are actively sought, and may be submitted at any time to NAVSEA 074.

Software engineering aspects of each phase of the acquisition process are explicitly covered in Section 1, from development of the Combat Systems Integration planning in the conceptual phase [p51:36] through computer program baselines in the Contract Design Phase [ibid:51] to computer programs in the Lead Ship Construction phase [ibid:59].

Section Two of Part Two [ibid:174] consists of over thirty pages of ECS software management lessons learned from one ship class acquisition. It covers basically all the topics covered in Section lll of this study, with particular attention to SHAPM/PARM relationships, PARM organization and management techniques, and the use of the Ship Project Directive system for intra-service contractual arrangements regarding software development [D7, D26, P14].

3. The R&D Plan [p50]

The September 1976 NAVSEA input to the August 1976 NAVMAT R&D plan outlines a plan to improve software acquisition through joint efforts in product development and process management methods. Product development is equivalent to what is normally considered as Computer Science; i.e., R&D to permit improved support software (e.g., compilers, monitors) and application software capabilities (adaptive data base inquiry methods, signal processing). Process management is associated with efforts to improve management of computer resource development in the ship acquisition process. It includes providing progressive project level practices and procedures for SHAPM or PARM, improved appraisal actions necessary in gaining product excellence in computer resources (acceptance test criteria, technical performance measurement), and improved ways of installing and checking out computer resources (acceptance test criteria, technical performance measurement), and improved ways of installing and checking out computer resources to reduce excessive testing.

A multitude of recommended R&D projects is listed and discussed, each tied to one of the DODI 5000.29 R&D objectives, with resource (time and dollars) estimates and an estimate as to whether near or long range payoff is expected.

A management plan and summary of expected results are included.

This is an ambitious program, and whatever portion of it survives the CSRDC validation and/or revision process due to complete shortly will undoubtedly influence the direction of Navy software management and software engineering in years to come.

4. The PATHWAY Program, sub-titled Product Quality Assurance for Shipboard Installed Computer Programs [P48]

This program has as its goal the development of the technology base to support creation of sound management practices and procedures which will provide full confidence and freedom from doubt of the quality of the software product within cost, schedule and other management constraints. The program has recently (October 1976) been incorporated in the NAVSEA portion of the NAVMAT R&D plan. PATHWAY has three interrelated concepts:

A. CONSERVE - (Control of Computer Resources) development of concepts and procedures for computer program resource allocation and use, including techniques for estimating, schedule control and transferability (portability) of software.

B. EXCEL - (Excellence in Substantiation, Validation and Certification) appraisal actions necessary in gaining product excellence in computer software. Included here are audits, analyses and overall computer software quality control. The NAVSEA definition of quality in a shipboard installed computer program is correctness (includes reliability) modificability and transferability.

C. PROV - (Programmed Validation of Major Ship Subsystem and System Installation and In-Service Maintenance Support) concepts for a new way of installing and checking out computer software. Specifically designed to attack the problem of excessive, uncoordinated and costly testing in ships.

Individual concept papers are provided under each of the three

major concepts as recommended areas for research and development effort. The program is viewed as dynamic, and in-process refinements are planned. SHAPM, PARM and other activity comments, recommendations, participation and support in the PATHWAY program would be welcomed as a valued part of the endeavor by the coordinator (NAVSEA 0744).

5. THE DSMC WORKSHOP [P47]

On 16 June 1976, participants from all services and OSD met for two days to surface and discuss critical problem areas impacting acquisition management and illuminate courses of action.

There was one finding not recorded in any of the previous study group, working group or contractor studies reported on in this paper, and it is one of key importance.

> .... many of the major problems..... are caused by a lack of understanding of software by personnel at the program review and approval level. Financial managers who do not understand the need for concurrent development of various aspects of support software recommend deletion or deferral of associated funding. [ibid, 1]

A reduced summary of the findings and recommendations follows:

Finding	Recommendation
1. Lack of discipline in specifying and validating requirements.	Require comprehensive analyses of systems requirements and defined software performance specifications prior to the decision to proceed with FSD.
2. Delay in development of support software causes delays in operational readiness of ECS.	Include individuals with software awareness in higher level budget reviews involving software, especially persons knowledgeable in: - training requirements - software maintenance and CM
3. Growth of systems with ECS creates new post deployment software support problems.	HQ and DOD recognition of the sizable investment needed for support of systems using ECS, particularly the need for centralized support facilities, and provision for the establishment of refined software support in a timely manner.

Finding	Recommendation
4. Shortage of software engineers in military and civil service.	Establish appropriate sub-specialty codes and develop career paths and incentives to attract and retain software engineers.
5. Proliferation of documentation standards within DOD.	SRWG of JLC develop a DOD standard for documentation.

The reader should remember that this workshop predates DODI 5000.29 by a few days.

SECTION V

CONCLUSIONS AND RECOMMENDATIONS

1. General Conclusions

After review and analysis of my investigations into the literature and interviews, I feel that I have achieved, at least in large part, the goals proposed for this study. I have added to my knowledge of embedded computer systems and ECS software engineering and software management to a considerable degree. Even if the remainder of the conclusions and recommendations below are of no value to anyone else, I feel I have reaped a great deal of personal benefit from this study.

As to the specific study project goals in Section 1.2, the suspicion that software management problems in Shipboard ECS are not really all that different from those in the large scale MIS, ADP, scientific and numerical control applications has been confirmed. The level of technical detail and expertise required of an ECS software engineer is certainly orders of magnitude away from that of an insurance company systems analyst or COBOL programmer, as he is much more intimately involved with the hardware and real time constraints, but the processes of planning, organizing, implementing, coordinating and controlling, and evaluation are very similar. Further, the management skills and methods of managing any complex design engineering effort apply directly to software development - in both cases, the product is an idea. The only difference is that we have all been brought up to read blueprints (the engineering product), but are ill at ease with program listings or flow charts, which are the graphic representation of the programming product.

Put yourself to this test - assume that you are a Naval Architect or Marine Engineer, acting as a SHAPM for a major combatant. If you were handed a computer program listing for calculating a sub-orbital trajectory or a blueprint for an integrated circuit universal asynchronous receiver transmitter interface plug-in module, would you be able (or expected) to detect and correct errors in either one? But, you could certainly be able to plan, budget and monitor to see that either was produced on time, within cost and within specification, and develop get well plans when something went wrong.

Which leads to the recommendations. I will try not to rehash the veritable plethora of recommendations which have come from all the committees, steering groups, councils, workshops, other industrial study projects, etc. In fact, one of the striking things to me about this study has been the discovery that everybody has known what's wrong and been making virtually the same recommendations since 1965; but, nobody seems to be getting any better. Maybe the recommenders are too busy studying and recommending, while the doers are out there stomping fires and trying to get rid of the alligators so we can get around to the swamp someday.

Since nothing is new under the sun, if these occasionally seem to overlap some other study's recommendations, I beg your indulgence.

2. Recommendations

Some of these recommendations are aimed at the individual manager who finds himself faced with managing software development, and some are aimed at the acquisition community in general.

A. Don't be afraid. As a (renegade?) member, I will break a sacred vow and reveal to you that there is a priesthood of computerniks whose life blood depends on persuading you of the mystique of the magical, sacred, unapproachable machine. These priests are prone to spout sacred oaths like "it must be done in machine language", or "I can't tell you what's wrong until I analyze this dump", all of which are designed to keep you away from the Almighty Machine which feeds and clothes them and protects their indispensability.

Romberg [P36] says there are two requirements for successful management of software production projects - a commitment to active participation in the process and successful experience in conventional management techniques. He is right (see Golub's Seventh Law). If you participate actively, you will pick up the little things you need to know by osmosis during the project.

To carry the priesthood analogy a little further - just as they were in medieval times, computer priests are often backward. In many cases, they have been the last to adopt automated scheduling techniques or any of the other marvels of management science that managers use constantly to manage non-computer projects. More than one Data Processing Director, with a multi-million dollar annual budget, uses bubbles on a bulletin board to schedule his computer runs of the PERT networks scheduling other

departments. And how many automated or semi-automated systems have you seen for handling all that non-program paperwork, like doing CPPS, CPPD, CPID and related alphabet soup bookkeeping at drastically increased productivity? Not many.

The moral is - don't let the priests buffalo you. It's quite likely that you're smarter than they are, and they're afraid you'll find it out.

B. Some recommended management techniques.

(0) PLAN, PLAN, PLAN

(1) Keep software development off the critical path. There are lots bigger problems in Shipbuilding, such as putting together a propulsion plant to pass the Propulsion Examining Board, or getting 400Hz power and dry air to the weapons system hardware. Plan to use your LBTS properly, and you can have most of the software tested before the equipment is installed on the Lead Ship.

(2) Please, somebody, bite the bullet and demand a software WBS in your contract - then accumulate an estimated vs. actual cost data base with some usable data. Who knows, maybe you can start a trend?

(3) Until that glorious day, here's a thumb rule (you'll have to figure out how to sell it to CAIG, or whoever). It's a combination of Brook's statement [P5] that the effort in a computer software task breaks out like this:

1/3 planning
1/6 coding
1/4 component test and early system test
1/4 integrated system test.

and Golub's 8th Law (see Appendix B).

The thumb rule:

a. Get the best estimate you can of how long it will take to code the problem and how much it will cost.

b. Multiply by six (Brooks).

c. Apply Golub's Eighth Law - since you're this far, you can assume you have a carefully planned project - double it!

C. Requirements and Specifications

(1) Make every effort to nail down the requirements early. Flaky requirements have caused more overruns than anything else.

(2) Decide what you want optimized in the programs, and put it in the specs. See Section III.1.E for details. This is a good place to call in your technical expert for help.

(3) Don't let the hardware people get away with a "leave it to the software" attitude to bail them out of technical problems. This is a feasible way of handling technical risks on occasion, but you should control it and know when and why you made the trade-off and the possible side effects. Use your Software Configuration Control Board to evaluate and make the recommendation to the full CCB (you will have both, won't you?).

(4) Don't forget the user. Because computer priests are happier that way, they spend more time talking to computers than they do to people. Often, this leads them to think that all people should communicate with machines in some fashion more acceptable to their deity, such as abstruse codes like "CC" or "31416D-4".

This can lead to horrendous problems. I just learned of an actual case in which twenty-six million dollars was lost because a programmer expected a warehouseman with a third-grade education to prepare a source document using the notation 23D3 to mean $23,000.00. The warehouseman didn't understand the D3, so he figured he's always be safe if he used ZERO! Errors accumulated for three months (in automatic) before the problem got big enough to detect, and now the organization is in a turmoil trying to recover the loss and retrain the warehouseman. Naturally, the programmer can't be found, and the program is nearly impossible to change.

Another major system has at least six ways of entering the date, and heaven help the poor user who uses MMDDYY when the program requires YMMDD - or tries to interpret 60228 in a report.

The moral is: Use your specifications to force the systems designers to make the system user-oriented. Inputs and outputs (man-machine interfaces) should be clear and comprehensible without recourse to a decoding manual. Make the computer decode it - a computer is cheaper than a people any day.

These are things that can be checked in a Design Review - and may shake up a priest or two when you have the temerity to raise the question.

D. Software Awareness Training for Managers

Excuse me for rehashing this one, but I've got to get on my soap-box for a minute.

(1) Beef up the embedded computer system software coverage in the Program Management Course at DSMC. I was appalled at the fear and ignorance shown by many of my contemporaries at this session when we went through the ECS portion of the Fundamentals of Program Management course.

These o-4's, O-5's, and GS-12 through 14's are our future, and they've got to be led past the fright stage and shown that computers and software can be managed and understood. I don't have a proposal for what to give up in the curriculum, but, with 5000.29 and the systems of the 80's staring us in the face, a way must be found.

(2) Use DSMC/DODCI (Department of Defense Computer Institute) for short courses for active or prospective PM's who can't spare the time for a long course. IBM runs (or did a few years ago) a five-day Executive Computer Course for their customers' top executives - its stated objective is to make them "unsnowable" by the priesthood. Surely, between DODCI and DSMC, we can do as well as IBM, but with more concentration on software and less on hardware.

(3) Start young. People who have grown up in the generation of television, hi-fi stereo, and C.B. are not as leery of computers as those of us who are in or near the 40+ bracket. It's much easier to crack the man-machine barrier before they're thirty. Computer technology is so all pervasive that money spent on computer-oriented short courses on our young officers, engineers and managers would not be wasted, even if they never were used as software engineers.

(4) Use the DSMC "Corporate Memory". One of the functions of the College is to serve as a center for management thought. The Individual Study Projects are not restricted to Software Management, but each class has several students both capable and interested in computers and software.

Part II of the References and Annotated Bibliography illustrates what has been done in the past. The current class (PMC 76-2) has fourteen computer-related studies in a class of 124 students.

There is a constant search for topics for individual study projects, and most students would be happy to participate in a meaningful study which would benefit one of the services at the same time.

Therefore, use the library to see if there is anything available pertaining to your particular problem, and make contact with the faculty to suggest projects that would help you.

E. Some Proposed Research Areas

So far, most of the Computer Science research seems to be centered on testing, automatic program validation, and quality assurance, which is well and good, because that's where 50% of the resources are expended in a programming project. However, let's not forget the other large drain, and one with potentially even higher payoff because of its potential for affecting 100% of the development cycle. I refer to the definition and design (system analysis and design) phase, which consumes about one-third of the resources.

We need research into automation of the analysis and design phase - possibly development of an automated standardized specification language, or automated system design languages (with built-in guarantees of testability), and certainly some development money spent in automation of the planning and documentation (not program documentation - I'm referring to generation of program specs from design and interface specs, or automated management paperwork generation). The whole process of con-

figuration identification, configuration control (except the decision making) and status accounting should be a piece of cake for an automated top-down design program librarian.

F. Organizational Recommendations

At first blush, the viewer gets the impression that there are too many players and no coach in the Shipboard Computer business, with our alphanumeric soup of involved codes - 00G, 01G, 651, 6513, MAT09Y, OP342, SEC 6170, SEC 6172, NELC, NUSC, FCDSSA and maybe some others. Perhaps a cleaner, more well defined organization might be a good thing, but after some consideration of the massive impact of all the DOD attention, all the studies, recommendations and new proposals thrown into the hopper in the past few months, and the potential for more to come, I feel that having the organization the way it is might be the best for right now, until things settle down a little. There is a good chance of making a premature move and locking in the wrong organizational structure, so I would recommend staying flexible. However, once the decision is made to develop some sort of software organization, I do recommend that it not be merged with the MIS/ADP side of the house. The program development concepts are similar, and the management techniques may be the same, but the mode of operation and the demands for management response are very, very different. Further, you would not want to create the condition where there is the remotest gleam in someone's eye that ECS should fall under the Brooks bill.

Meanwhile, NAVSEA should continue to publicize the existence and actions of the NAVSEA Steering Group, staff it properly, and encourage

all codes to provide input, guidance, management attention, and support to the PATHWAY program, the R&D program, REEF POINTS updates and other efforts of the Steering Group.

APPENDIX A

INTERVIEW QUESTIONNAIRE

I. Planning - Preliminary Planning & Cost Analysis

Systems Analysis & Design

1. General

How and to what level of detail were software requirements developed prior to contracting? Who managed it, SHAPM or PARM?

How clearly and early were requirements finalized?

What policies, procedures, regulations or standards were available to you? Did you opt to use any? Were they useful?

Was there any controlled effort to use existing software modules? Was it worthwhile?

Were Hardware-Software trade-offs made before software requirements were developed?

What factors caused the greatest problems? (e.g. - requirements, interfaces, cost?)

Do you think management of software development can be handled in the same way as for hardware?

How would you evaluate your SHAPM regarding the timeliness and completeness of software management?

If you could do it over, what would you change?

2. WBS

Was a Work Breakdown Structure used in developing functional requirements and for planning, scheduling and estimating. To what level in the SHAPM?

3. Cost Estimating

What process was used for estimating costs?

How detailed? How often were they refined/revised?

Was there any NAVSEA/NAVMAT "corporate memory" available?

Were any elements overlooked that you would include next time?

Can you given an estimate of how your return costs compared to the original estimates?

How about software costs compared to hardware costs?

4. Specifications

Were Higher Order Languages or programming style (i.e., structured, modular) specified?

What type of documentation was called out in the CDRL?

Was it delivered satisfactorily?

Were the B5(CPDS) and C5(CPPS) specifications of MIL STD 490 usable? Did you note any deficiencies?

How about the Configuration Management Specs (MIL STD 480-483)?

Who had integration responsibility?

Were programs specified as deliverable CI's?

Did you have a liquidated damages clause?

III. Implementation - Program design, coding and debugging

1. *Hardware interface*

What was your mainframe?

What as the quality of the support software (utilities, monitor, maintenance routines)?

Did hardware limitations compound the software development problem?

Were there any conflicts because of hardware-software development in the same time frame?

2. Programming

Did programming/coding start prior to CDR?

Were programmers encouraged to optimize particular parameters (i.e., execution time, minimum core, system overhead, etc.)?

Was the contractor's definition of "optimum" the same as yours?

Any problems with software rights? Proprietary contractor information?

III. Evaluation - System Test and Integration
Installation and Turnover

1. Test

What was your test philosophy? Who controlled testing - SHAPM in-house or an independent test agent?

How was software tested? What procedures (stubs, simulation, LBTS)?

Did software testing receive the same attention as hardware?

Were any work-arounds used or allowed?

2. Integration

Who had system integration test responsibility?

Was it effective?

Did any major systems integration problems surface during software testing?

5. Documentation

How was documentation controlled?

Was the quality adequate?

Did you have guidance and standards for deliverables or have to develop your own?

Were the standards adequate?

6. Coordination

Were you able to control the scope after contract award, or did the requirement continue to change?

Did you have any security or classification problems?

Were contractor requirements clearly spelled out in the statement of work without reference to in-house government documents not available to him?

Did you have any prime/sub-contractor problems?

GFE (i.e., test equipment, hardware, computer time. etc.) problems?

If you had it to do over, would you make any changes? Do you think DODI 5000.29 will have any material effect on your operation?

V. Organization

1. Management Organization

Was there a separate software management function in the SHAPM or PARM?

How about the converse (software problems during integration testing)?

Did you experience any serious effects on the acquisition process (cost, schedule, performance) as a result of integration testing problems?

3. Quality assurance, reliability and maintainability

How did you manage the QA function?

Was there any impetus for acceptance of the system when the hardware was complete, regardless of the software status? If so, from where?

4. Installation & Turnover

How was crew training handled?

Was it successful?

What changes would you make?

IV. Coordination and Control

1. Techniques

Were any modern management science or systems engineering techniques used in software development (PERT/CPM, WBS, etc.)?

What types of milestones were established?

Were they tied to deliverables?

Did you receive product specs. in time for preparation for PDR and CDR?

Was your schedule realistic? Did the contractor complete on time?

2. Monitoring

How were cost, schedule and performance monitored?

How often? By whom?

Was tracking tied to WBS, network or cost accounts?

Was the schedule kept current? Deviations controlled/logged?

Were interim design reviews held to be sure contractor was ready for the formal PDR, CDR?

What level of detail and frequency?

Do you think your system is worthy of broad adoption?

Did you observe any signs of buried indirect costs?

3. Interface Control

What form of interface management was used?

CPIS/CPID?

Who had cognizance of interface control?

Was it strictly controlled and documented? Formal Board?

Has it been satisfactory - what problems have occurred?

4. Configuration/Change Control

Were CM Baselines used?

Was there a formal system for keeping them current?

What processes/procedures were adopted for authorization, control and documentation of changes? CCB?

When in the development cycle were they adopted?

Was the contractor required to provide a software development plan (DID E-695/ESD)?

Did you have separate hardware and software contractors?

Was there a "prime" for systems development, or a design agent?

Did the same contractor develop the CPDS, CPPS and the programs themselves?

2. Personnel Capabilities

How would you rate top management regarding understanding of software? How about in-house talent below the top?

Was there any training in software awareness for PM's?

Were competent personnel available in the PMO or on call in the functional organization?

Do you think there should be a career field for GS or Military software management specialists (as opposed to programmers/ computer scientists)?

3. Coordination

Did you experience any organizational/management problems in review of the specifications? From the user? Functional codes? Top Management?

How were relations with the contracting officer - was the purchase function technically competent regarding software?

Were you able to pull together a qualified SSEB? What was its composition?

Was the selected contractor really responsive? Capable?

Did the RFP let in any dogs?

4. Miscellaneous

Did either the SHAPM, PARM or contractor establish a program librarian? Was it useful?

Do you know of a software management forum within NAVSEA/NAVMAT/DOD for exchange of experience, problems, and successful methods?

Do you think this is necessary for development of corporate memory?

On what elements of software management should they concentrate?

APPENDIX B

GOLUB'S LAWS OF COMPUTERDOM

(WITH APOLOGIES TO MURPHY)

Law #1 - No major computer project is ever installed on time, within budget, with the same staff that started it, nor does the project do what it is supposed to...it is highly unlikely that yours is going to be the first.

Corollary #1 - The benefits will be smaller than initially estimated, if estimates were made at all.

Corollary #2 - The system finally installed will be installed late and won't do what it is supposed to.

Corollary #3 - It will cost more but it will be technically successful.

Law #2 - One advantage of fuzzy project objectives is that they let you avoid embarrassment in estimating the corresponding costs.

Law #3 - The effort required to correct course increases geometrically with time.

Corollary #1 - The longer you wait (to define objectives) the harder it gets.

Corollary #2 - After installation is too late.

Corollary #3 - Do it now.

Law #4 - Purposes as understood by the purposer will be seen differently by everyone else.

Corollary #1 - If you explain so clearly that nobody could possibly misunderstand, somebody will.

Corollary #2 - If you do something that you are sure will meet everybody's approval, somebody won't like it.

Law #5 - Only measurable benefits are real. Intangible benefits are not measurable. Therefore, intangible benefits are not real.

Law #6 - Anyone who can work effectively on a project part time certainly doesn't have enough to do now.

Corollary #1 - If his boss won't give him a full-time job you shouldn't either.

Corollary #2 - If he has a time conflict, his boss' work won't suffer.

Law #7 - The greater the project's technical complexity, the less you need a technician to manage.

Corollary #1 - Get the best manager you can; he'll get the technician.

Corollary #2 - The reverse is almost never true.

Law #8 - A carelessly planned project will take three times longer to complete than expected. A carefully planned project will only take twice as long.

Law #9 - If anything can go wrong, it will.

Corollary - If nothing can possibly go wrong, it will anyway.

Law #10 - When things are going well, something will go wrong.

Corollary #1 - When things just can't get any worse, they will.

Corollary #2 - When things appear to be going better, you have overlooked something.

Law #11 - Project teams detest weekly progress reporting because it so vividly manifests their lack of progress.

Law #12 - Projects progress quickly until they are 90 percent complete, and they they remain 90 percent complete forever.

Law #13 - If project content is allowed to change freely, the rate of change will exceed the rate of progress.

Law #14 - If the user does not believe in the system, he will develop a parallel system...Neither system will work very well.

Law #15 - Benefits achieved are a function of the thoroughness of the post-audit check.

Corollary #1 - The independent post-audit provides the team with a powerful incentive to deliver a good system on schedule.

Law #16 - No law is immutable.

- Groobey, John A. Data Management
September, 1975 pp28-32

References and Annotated Bibliography

Part 1. Directives, Standards and Official Documents

D1.	DODINST 5000.1	Acquisition of Major Defense Systems
D2.	DODINST 5000.2	The Decision Coordinating Paper (DCP) and the Defense Systems Acquisition Review Council (DSARC)
D3.	DODINST 5000.3	Test and Evaluation
D4.	DODINST 5000.26	Defense Systems Acquisition Review Council (DSARC)
D5.	DODINST 5000.29	Management of Computer Resources in Major Defense Systems
D6.	OPNAVINST 4130.1	Configuration Management of Software in Surface Ship Combat Systems
D7.	NAVSEAINST 7000.3	Ship Project Directive System
D8.	NAVSEAINST 9060.4	Ship Acquisition Process
D9.	NAVSEAINST 9060.5	Ship Acquisition and Logistic Support, Combat System Design Requirements and Combat System Operational Design
D10.	NAVMATNOTE 5420	NAVMAT Council for R&D in Computer Sciences (SRDC)
D11.	MIL-STD 480	Configuration Control - Engineering Changes, Deviations and Waivers
D12.	MIL-STD 481	Configuration Control - Engineering Changes, Deviations and Waivers (Short Form)
D13.	MIL-STD 482	Configuration Status Accounting Data Elements and Related Features
D14.	MIL-STD 483 (USAF)	Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs
D15.	MIL-STD 490	Specification Practices

D16.	MIL-STD 499	System Engineering Management
D17.	MIL-STD 499A (USAF)	Engineering Management
D18.	MIL-STD 881	Work Breakdown Structures for Defense Material Items
D19.	MIL-STD 885A	Procurement Data Package
D20.	MIL-STD 1521 (USAF)	Technical Reviews and Audits for Systems, Equipment and Computer Programs
D21.	MIL-STD 1697 (USN) (Draft-has no official status)	Tactical Software Development
D22.	DA Pamphlet 5-4-6	Work Scheduling Handbook
D23.	NASA TM X-65810	Standards Guide for Space and Earth Sciences Computer Software (NTIS N72-16148)
D24.	NAVMAT P5240	Cost/Schedule Control Systems Criteria Implementation Guide
D25.	NAVORD WS-8506 (Rev. 1)	Requirements for Digital Computer Program Documentation
D26.	NAVSEA 0900-080-0910	Ship Project Directive System Manual
D27.	NAVSHIPS 0967-011-0090/11	Specifications for Digital Computers Program Documentation
D28.	TB 18-19-1-7	MIS Handbook of ADP Resource Estimating Procedures
D29.	TM 38-760 (ARMY)	System Engineering Summary

Part II. Defense Systems Management College - Individual Study Projects

11. Andres, A.W. Estimating Computer Software Development Costs
PMC 75-1 May 1975
Title is self-explanatory. A thorough summary.

12. Bucciarelli, M.A. Technical Performance Measurement for Computer Software Development Programs
PMC 74-1 May 1974
Software Engineering and cost, schedule and performance monitoring methods.

13. Capps, L.R. Software Management and the Testing of Weapons Systems that Contain an Embedded Computer System
PMC 75-2 Nov 75
Overview of DOD Software Management program, including limitations related to Test and Evaluation, and discussion of considerations that must be emphasized.

14. Drabant, T.M. An Introduction to Management of Weapons System Software Development
PMC 75-1 May 75
A teaching paper for Program Managers needing information on software management by a professional in the field. Clear, well written.

15. Dunham, D.C. Navy Airborne Tactical Software Management Policy: Review and Evaluation
PMC 75-2 Nov 75
Review of policies pertaining to configuration management of airborne tactical software.

16. Ervin, J.R. Testing of Automatic Command and Control Systems
PMC 75-2 Nov 75
Layman's overview of C&C systems testing methodology.

17. Farnan, D.C. Reliable Computer Software: What It is and How to Get It
PMC 75-2 Nov 75
Review of techniques such as Top Down, egoless programming, etc. which can improve software design and increase reliability.

18. Findley, R. Computer Software Development Costs, Predictable or Not?
PMC 74-1 May 74
Review of methods of estimating software costs and impact on Program Manager.

19. Gurskis, J.J. Configuration Control of Surface Navy Weapon System Software: Post Development
PMC 75-2 Nov 75
Contains a draft configuration control plan for an embedded computer system.

110. Mielo, H.E. Digital Computers in U. S. Naval Combat Systems
PMC 73-1 May 1973
Considers man-machine interface. Software management not a major consideration.

111. Pontius, H.E. Life Cycle Guidelines for Weapons System Software Management
PMC 76-1 May 1976
Presentation of current DOD and USAF software management policies. Contains numerous recommendations for use by Program/System Manager in managing a software development effort. Large bibliography.

112. Robinson, T.J. Overview of Software Acquisition for Army Tactical Data Systems
PMC 75 2 Nov 75
Software management in the TACFIRE, Missile Minder and Tactical Operations Systems. Reviews current guidance and standards as applied to software.

113. Walters, L.R. Computer Resources Acquisition and Support for Air Force Weapons Systems
PMC 75-1 May 75
Review of USAF procedures in terms of CM, data rights and support. Emphasizes systems engineering processes applied to software engineering.

114. Zabriskie, R.J. Development of Weapon Systems Computer Programs: Guidelines for Controlling During FSD
PMC 75-2 Nov 75
Excellent reference for software development, related to Shipboard ECS, AEGIS in particular.

Part III. Text and Reference Books

T1. Adams, E.B. — Management of Information Technology: Case Studies
New York: Petrocelli/Charter, Inc. 1975
A compilation of cases in ADP/MIS management which illustrates the similarity of management issues.

T2. Archibald, R.D. and Villoria, R.L. — Network Based Management Systems (PERT/CPM)
New York: John Wiley & Sons, Inc. 1967
Textbook on network based methods and applications.

T3. Cronstedt, V. — Engineering Management and Administration
New York: McGraw-Hill Book Co., 1961
Text and reference book for use by engineering department managers and administrators.

T4. Dijkstra, E.W. — Structured Programming
New York: Academic Press 1972
Textbook of Structured Programming techniques, Dijkstra's work has been seminal in this development.

T5. Hetzel, W.C. ed. — Program Test Methods
New Jersey: Prentice-Hall, Inc. 1973
Proceedings of Symposium of Computer Program Test Methods held at University of North Carolina, Chapel Hill, N.C. in June 1972. Topics include program testing, standards, and quality measurement.

T6. Hice, G.F., Turner, W.S. and Cashwell, L.F. — System Development Methodology
Amsterdam: North Holland Pub. Co. 1974
An explicit exposition of the software system development process. Very detailed. Includes suggested activity networks for all phases of the development process.

T7. Hicks, T.G. — Successful Engineering Management
New York: McGraw-Hill Book Co. 1966
Generalized management techniques oriented to the engineer and scientist for effective and profitable direction of the engineering function.

T8. Knuth, D.E. The Art of Computer Programming, Vol 1/Fundamental Algorithms
Massachusetts: Addison-Wesley Publishing Co, 1968

T9. Metzger, P.W. Managing a Programming Project
New Jersey: Prentice-Hall, Inc, 1973
Short, clear, lucid Bible for software management. A classic in the field.

T10. Ridge, W.J. and Johnson, L.E. Effective Management of Computer Software
Illinois: Dow Jones-Irwin, Inc, 1973
Concepts of Value Engineering applied to the computer software management process. Includes Design-to-Cost, Motherhood and Apple Pie.

T11. Rubin, M.L. ed. Handbook for Data Processign Management
New Jersey: Auerbach Publishers, Inc. 1971
A six volume set of reference books for ADP management.

T12. Sanders, D.H. Computers and Management
New York: McGraw-Hill Book Co. 1974
Management oriented introduction to computer utilization. Includes selected readings from the Data Processing and Management literature.

T13. Szweda, R.A. Information Processing Management
New Jersey: Auerbach Publishers, Inc, 1972
Standard college level text on Data Processing management. Strong on project scheduling.

T14. Warnier, J.D. The Logical Construction of Programs (translated to English in 1974. Not listed in Books in Print 1975)
Description of top-down basis for a complete approach to structured analysis and design, rather than considering structured programming alone. Widely used outside U.S. May become landmark in software engineering.

T15. Watson, R.W. Timesharing System Design Concepts
New York: McGraw-Hill Book Co, 1970
Concepts for design of large-scale interactive hardware and operating systems. Aspects of response time, reliability and fault tolerance correspond directly to real-time monitor/executive systems.

T16. Weber, M. The Theory of Social and Economic Organization (translated by Henderson, A.M, and Parsons, T.) New York: The Free Press of Glencoe 1964 p.328

Part IV. Publications

P1. Ascani, F. and Low , A. — Fundamentals of Program Management. Virginia: Defense Systems Management College, lecture notes (unpublished), 1976

P2. Asch, A. et al — DOD Weapon Systems Software Acquisition and Management Study, Vol. 1, MITRE Findings and Recommendations. Virginia: The MITRE Corporation, 1975

P3. Baker, F.T. — Chief Programmer Team Management of Production Programming, IBM Systems Journal 11, No. 1, 1972

P4. Bennett, J. — Programmer Productivity. Proceedings of the AIIE Software Conference, July 1976, pp 139.

P5. Brooks, F.P. — The Mythical Man-Month. Datamation 20, No. 12, December 1974, pp 45-52.

P6. Brown, J.R. — Controlling and Measuring Software Quality. Proceedings of the AIIE Software Conference, July 1976, pp 633-644.

P7. Chapin, N., House, R., McDaniel, N. and Wechtel, R. — Structured Programming Simplified. Computer Decisions, June 1974, pp 28-31.

P8. Clapp, J.A. — Software Engineering: Problems and Future Developments. Massachusetts: The MITRE Corporation, November 1974. AD A003422

P9. Clapp, J.A. and LaPadula, L.J. — Engineering of Quality Software Systems. Massachusetts: The MITRE Corporation, January 1975. AD A007766

P10. Cooper, J.D. — Increased Software ransferability Dependent on Standardization Effort. Defense Management Journal 11, No. 4, October 1975, pp 19-22.

P11. Curry, G.D. — Software Procurement Specifications. Proceedings of the AIIE Software Conference, July 1976, pp 721-741.

P12. DD DR&E (T&E) — Test and Evaluation Guidelines for Command and Control Systems
Washington: Department of Defense, 2 April 1974, p. 9.

P13. Defence Science Board, ODDR&E — Report of Task Force on Test and Evaluation
Washington: Department of Defense, 2 April 1974, p. 13.

P14. DeMarco, I.J. — Shipboard Installed Computer Program Acquisition - A new Area for Ship Project Directive Application
NAVSEA Journal 25, No. 3, March 1976, pp 12-17.

P15. DeRoze, B.C. — An Introspective Analysis of DOD Weapon System Software Management
Defense Management Journal, 11, No. 4, October 1975, pp 2-7.

P16. Gill, G.W. and Jensen, A.P. — Management of Computer Programming Part 1: Practices and Problems
Georgia: Georgia Institute of Technology, 1969
NTISPB 185470 (see p.17)

P17. Gill, G.W. and Jensen, A.P. — Management of Computer Programming Part 2: Case Studies
Georgia: Georgia Institute of Technology, 1970
NTIS PB 185470
Case studies and analysis of management procedures in non-Defense systems.

P18. Groobey, J.A. — Maximizing Return on ADP Investments
Data Management, September 1972, pp 28-32.

P19. Hecht, H. — Fault Tolerant Software for Spacecraft Applications
California: Aerospace Corporation, December 1975. AD A022068

P20. Jones, M.N. — HIPO for Developing Specifications
Proceedings of the AIIE Software Conference, July 1976, pp 743-751.

P21. La Bolle, V. — Critical Management Points
California: Systems Development Corporation, February 1965. AD 611867

P22. Larr, L., La Bolle, V. and Willworth, N.E. — Planning Guide for Computer Program Development. California: Systems Development Corporation 1965. AD 734358. A detailed guide for software management planning developed for the Naval Command Systems Support Activity.

P23. Lipow, M. — Measurement of Computer Software Reliability. Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group (SRWG Report), Volume II: Supporting Technical Information. November 1975, contributed paper, pp 123-143.

P24. McCarthy, R. — Applying the Technique of Configuration Management to Software. Defense Management Journal 11, No. 4, October 1975, pp 23-28.

P25. Manley, J.H. — Embedded Computer System Software Reliability. Defense Management Journal 11, No. 4, October 1975, pp 13-18.

P26. Mathis, N.S. and Willworth, N.E. — Software Milestone Measurement Study. California: Systems Development Corporation, 1973. AD 775305.

P27. Moore, B.B. — Software Requirements Specifications. Proceedings of the AIIE Software Conference, July 1976, pp 711-719.

P28. Morgan, J.H. and Lightman, M.S. — System for Developing Systems. Proceedings of the AIIE Software Conference, July 1976, pp 645-649.

P29. Orr, K.T. — Beyond Structured Programming. Infosystems, October 1975, pp 49-50.

P30. Orr, K.T. — Structured Programming: Not a Fad. Infosystems, November 1975, pp 36-40.

P31. Peters, L.J. and Tripp, L.L. — Is Software Design "Wicked"? Proceedings of the AIIE Software Conference, July 1976, pp 495-499.

P32. Piligian, M.S., Bashaw, C.J., and Pokorney, J.L. — Configuration Management of Computer Program Contract End Items. Massachusetts: Electronic Systems Division, AFSC, USAF, January 1968. AD 666652.

P33. Pilgian, M.S. and Pokorney, J.L. — Air Force Concepts for the Technical Control and Design Verification of Computer Programs. Massachusetts: Electronic Systems Division, AFSC, USAF, April 1967. AD 652209

P34. Randell, B. — System Structure for Fault Tolerance. Proceedings, 1975 International Conference on Reliable Software, p. 437.

P35. Richardson, D.R. — The People Side of Top-Down. Infosystems, July 1975, pp 34-35.

P36. Romberg, B.W. — Managing Software: It Can Be Done. Infosystems, November 1972, pp 42-43.

P37. Schneidewind, N.F. — A Methodology for Software Reliability Prediction and Quality Control. California: Naval Postgraduate School, November 1972. AD 754377

P38. Spaulding, K.B. and Johnson, A.F. — Management of Ship Design at the Naval Ship Engineering Center. Naval Engineer's Journal 88, No. 1, February 1976, pp 27-44.

P39. Sternberg, C.D. — The Structured Approach: A Fifth Dimension. Infosystems, January 1976, pp 47-48.

P40. Vote, F.W. — Multiprogramming Systems Evaluated Through Synthetic Programs. Massachusetts: Massachusetts Institute of Technology-Lincoln Laboratory 1973. AD 771798

P41. Ward, J.A. and Geller , G. — Report for Commander, NAVSEA on Estimating Costs of Digital Computer Systems (internal NAVSEA report). Washington: Naval Sea Systems Command, December 1975.

P42. Weinwurm, G.F. and Zagorski, H.J. — Research into the Management of Computer Programming: A Transitional Analysis of Cost Estimation Techniques. California: Systems Development Corporation, 1965.

P43. Zempolich, B.A. — Effective Software Management Requires Consideration of Many Factors. Defense Management Journal II, No. 4, October 1975, pp 8-12.

P44. ---- DOD Weapon Systems Software Management Study
Maryland: The Johns Hopkins University Applied Physics Laboratory, 1975.

P45. ---- Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group
Washington: Department of Defense, November 1975 (two volumes) AD A018881, AD A018882

P46. ---- Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980s (CCIP-85). Vol. IV Technology Trends: Software
California: Space and Missile Systems Organization, USAF, October 1973. AD 919267L

P47. ---- Management of Software Acquisition for Embedded Computer Systems, Report on DSMC Workshop
Virginia: Defense Systems Management College, June 1976.

P48. ---- Pathway Program - Product Quality Assurance for Shipboard Installed Computer Programs
Naval Sea Systems Command tentative working papers (unpublished - distribution limited to U. S. Government agencies only) June 1976.

P49. ---- Proceedings of the American Institute of Industrial Engineers Software Conference Chapter 2: Software Costs
California: Management Education Corporation, July 1976, pp 165-355. Summary of latest thoughts on estimating and controlling software costs.

P50. ---- Research and Development Program for the Management of Computer Resources in Navy Systems
Naval Material Command tentative working papers (unpublished - distribution limited to U. S. Government agencies only) September 1976.

P51. ---- Ship Acquisition REEF POINTS
Washington: Naval Sea Systems Command, September 1976.

P52. ---- SHAPM Interviews.